Searching PAJ 1/2 ページ

PATENT ABSTRACTS OF JAPAN

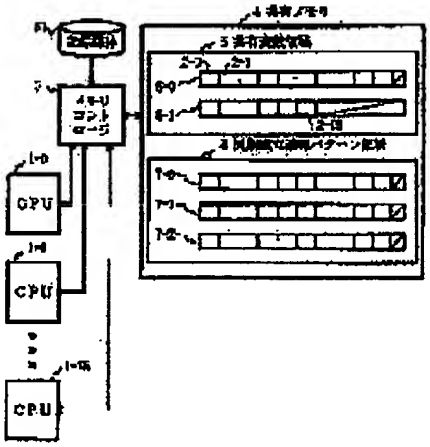
(11)Publication number : 2001-043203
(43)Date of publication of application : 16.02.2001

(51)Int.Cl. G06F 15/177

(21)Application number : 11-214741 (71)Applicant : NEC CORP
(22)Date of filing : 29.07.1999 (72)Inventor : ARAKI HIROYUKI

(54) BARRIER SYNCHRONIZING METHOD AND RECORDING MEDIUM STORING PROGRAM FOR BARRIER SYNCHRONISM

(57)Abstract:
PROBLEM TO BE SOLVED: To reduce the size of a shared variable for barrier synchronism and to confirm whether barrier synchronism is established or not with a little arithmetic quantity in the case of barrier synchronism between plural CPUs composing a parallel computer.
SOLUTION: Shared variables 2-0 to 2-15 for barrier synchronism are individually allocated to respective CPUs 1-0 to 1-15 composing the parallel computer. Each time the execution of a present CPU reaches a barrier part, each CPU 1-i ((i) is 1 to 15) updates the value of a shared variable 2-i corresponding to the present CPU to a value showing the next generation according to the ring- shaped transition rules of three generations, detects whether all the shared variables corresponding to the other CPU are matched with the generation before the update by comparing the stream of values of the plural shared variables with prescribed synchronism establishment confirmation patterns 7-0 to 7-2 and discriminates barrier synchronism establishment when all the shared variables corresponding to the other CPU are not matched with the generation before the update.



LEGAL STATUS

[Date of request for examination]	20.06.2000
[Date of sending the examiner's decision of rejection]	
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	3304928
[Date of registration]	10.05.2002
[Number of appeal against examiner's decision of rejection]	

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号
特開2001－43203
（P2001－43203A）
(43)公開日　平成13年２月16日(2001.2.16)

(51)Int.Cl.'	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｆ　15/177	６８１	Ｇ０６Ｆ　15/177	６８１Ｃ　５Ｂ０４５

審査請求 有　請求項の数12　ＯＬ　（全 13 頁）

(21)出願番号　特願平11－214741

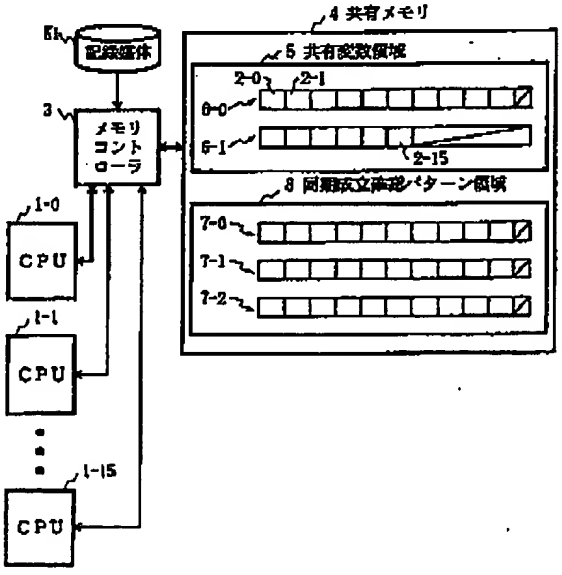
(22)出願日　平成11年７月29日(1999.7.29)

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(72)発明者　荒木　宏之
東京都港区芝五丁目７番１号　日本電気株式会社内
(74)代理人　100088959
弁理士　境　廣巳

Ｆターム(参考)　5B045 CC06 GG11

(54)【発明の名称】　バリア同期方法およびバリア同期用プログラムを記録した記録媒体

(57)【要約】
【課題】　並列計算機を構成する複数のＣＰＵ間でバリア同期をとる際、バリア同期用の共有変数のサイズを小さくでき、バリア同期が成立したか否かを少ない演算量で確認できるようにする。
【解決手段】　並列計算機を構成する各ＣＰＵ1-0 ～1-15にバリア同期用の共有変数2-0 ～2-15を個別に割り当てる。各ＣＰＵ1-i(i=1 ～15) は、自ＣＰＵの実行がバリア部分に達する毎に、自ＣＰＵに対応する共有変数2-iの値を３世代のリング状遷移規則に従って次の世代を示す値に更新し、他ＣＰＵに対応する共有変数の全てが前記更新前の世代と一致しないかどうかを、複数の共有変数の値の列と所定の同期成立確認パターン7-0 ～7-2とを比較することにより検出し、他ＣＰＵに対応する共有変数の全てが前記更新前の世代と一致しない状態となったときにバリア同期成立と判定する。



1

【特許請求の範囲】

【請求項1】 並列計算機を構成する各CPUにバリア同期用の共有変数を個別に割り当て、各CPUにおいて、自CPUの実行がバリア部分に達する毎に、自CPUに対応する共有変数の値を3世代のリング状遷移規則に従って次の世代を示す値に更新し、他CPUに対応する共有変数の全てが前記更新前の世代と一致しないかどうかを、複数の共有変数の値の列と所定の同期成立確認パターンとを比較することにより検出し、他CPUに対応する共有変数の全てが前記更新前の世代と一致しない状態となったときにバリア同期成立と判定することを特徴とするバリア同期方法。

【請求項2】 並列計算機を構成する複数のCPU間でバリア同期をとる方法において、

前記CPUに1対1に対応する共有変数の全てを、同じ同期世代を示す値に初期化する第1のステップと、

バリア部分に実行が達した前記CPUのそれぞれにおいてバリア同期処理を実行する第2のステップとを含み、

前記バリア同期処理は、(a)第1世代の次の世代を第2世代、第2世代の次の世代を第3世代、第3世代の次の世代を第1世代とする3世代のリング状遷移規則に従って、自CPUに対応する前記共有変数の値を、その値が示す世代の次の世代を示す値に更新するステップと、(b)他CPUに対応する共有変数の全てが自CPUに対応する前記更新前の共有変数の値と一致しないという条件が成立するまで待ち合わせ、前記条件が成立したときバリア同期の成立と判定するステップとを含むことを特徴とするバリア同期方法。

【請求項3】 前記ステップbは、前記更新前の共有変数の値を複数個並べた同期成立確認パターンと、他CPUに対応する共有変数の値を同じ個数並べたパターンとを比較する処理を含むことを特徴とする請求項2記載のバリア同期方法。

【請求項4】 予め作成された3種類の前記同期成立確認パターンの中から適切なものを選択して使用することを特徴とする請求項3記載のバリア同期方法。

【請求項5】 前記共有変数のビット数を3ビットとし、前記CPUから1度でリード、ライトできる8×m(mは1以上の整数)ビットの記憶領域に8×m÷3の商に等しい個数の共有変数を格納し、前記同期成立確認パターンとして8×mビットのパターンを使用することを特徴とする請求項3または4記載のバリア同期方法。

【請求項6】 前記全ての共有変数を全CPUで共有される共有メモリ上に格納するようにしたことを特徴とする請求項5記載のバリア同期方法。

【請求項7】 前記全ての共有変数を、他CPUから遠隔メモリアクセス可能な前記それぞれのCPUのローカルメモリ上に格納するようにしたことを特徴とする請求項5記載のバリア同期方法。

【請求項8】 自CPUに対応する前記共有変数の値の更新をアトミックな論理演算を用いて行うことを特徴とする請求項6または7記載のバリア同期方法。

【請求項9】 前記共有変数のビット数を8ビットとし、前記CPUから1度でリード、ライトできる8×r(rは2以上の整数)ビットの記憶領域にr個の共有変数を格納し、前記同期成立確認パターンとして8×rビットのパターンを使用することを特徴とする請求項3または4記載のバリア同期方法。

【請求項10】 前記全ての共有変数を全CPUで共有される共有メモリ上に格納するようにしたことを特徴とする請求項9記載のバリア同期方法。

【請求項11】 前記全ての共有変数を、他CPUから遠隔メモリアクセス可能な前記それぞれのCPUのローカルメモリ上に格納するようにしたことを特徴とする請求項9記載のバリア同期方法。

【請求項12】 並列計算機を構成する複数のCPU間でバリア同期をとるためのプログラムを記録した記録媒体であって、

前記それぞれのCPUに、

前記CPUに1対1に対応する共有変数の全てを、同じ同期世代を示す値に初期化する第1のステップ、

バリア部分に実行が達した前記CPUのそれぞれにおいてバリア同期処理を実行するステップであって、(a)第1世代の次の世代を第2世代、第2世代の次の世代を第3世代、第3世代の次の世代を第1世代とする3世代のリング状遷移規則に従って、自CPUに対応する前記共有変数の値を、その値が示す世代の次の世代を示す値に更新するステップと、(b)他CPUに対応する共有変数の全てが自CPUに対応する前記更新前の共有変数の値と一致しないという条件が成立しているか否かを、前記更新前の共有変数の値を複数個並べた同期成立確認パターンと他CPUに対応する共有変数の値を同じ個数並べたパターンとを比較することで検出し、前記条件が成立していないときは前記条件が成立するまで待ち合わせ、前記条件が成立したときバリア同期の成立と判定するステップとを含む第2のステップ、

を実行させるバリア同期用プログラムを記録した記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は複数のプロセッサからなる並列計算機における同期技術に関し、特にバリア同期に関する。

【0002】

【従来の技術】同期方式の一種であるバリア同期は、並列計算機においては複数のCPUで処理のタイミングを合わせるために用いられる。従来のバリア同期方法の一例が、特開平6-243110号公報に記載されている。

【0003】同公報に記載された従来のバリア同期方法

3

では、複数のＣＰＵで共有されるメモリ上に、各ＣＰＵに1対1に対応する共有変数の領域を確保し、最初に全ての共有変数を数値“０”に初期化しておく。各ＣＰＵの実行がバリア部分に達すると、各ＣＰＵは自己に対応する共有変数を１増加させた後、自己以外の他のＣＰＵに対応する共有変数の値の全てが、自己のＣＰＵに対応する共有変数の値以上になるまで待ち合わせを行い、その条件を満足したときに、バリア同期の成立と判定する。以後、各ＣＰＵはバリア部分に達する毎に同様の動作を繰り返す。これにより、各共有変数の値は1回のバリア同期が成立する毎に1ずつ増加していく。

【０００４】

【発明が解決しようとする課題】上述した従来のバリア同期方法によれば、ＣＰＵに1対1に対応する共有変数の領域を確保したことにより、１つの共有変数を複数のＣＰＵで排他的に定義、参照してバリア同期をとる方法に比べて、ロック制御が不要になり、その分だけバリア同期処理に要する処理時間を短縮することができる。しかしながら、かかる従来技術においても、未だ解決すべき課題が残されている。

【０００５】その1つは、各ＣＰＵに対応する共有変数の値はバリア同期が成立する毎に1ずつ増加していくため、何度となく繰り返されるバリア同期を毎回支障なく行えるようにするためには、共有変数の最大ビット長を充分に大きくしなければならないことである。

【０００６】他の課題は、各ＣＰＵにおいて他のＣＰＵに対応する共有変数の値の全てが自己のＣＰＵに対応する共有変数の値以上であることを調べる必要があり、そのためには、各ＣＰＵにおいて他のＣＰＵの台数分の比較演算を実行しなければならないことである。

【０００７】そこで本発明の目的は、各ＣＰＵに対応する共有変数のサイズを小さくできるようにすることにある。

【０００８】また本発明の別の目的は、バリア同期が成立したか否かを少ない演算量で確認できるようにして、バリア同期処理の高速化を図ることにある。

【０００９】

【課題を解決するための手段】本発明は上記の目的を達成するために、並列計算機を構成する各ＣＰＵにバリア同期用の共有変数を個別に割り当て、各ＣＰＵにおいて、自ＣＰＵの実行がバリア部分に達する毎に、自ＣＰＵに対応する共有変数の値を3世代のリング状遷移規則に従って次の世代を示す値に更新し、他ＣＰＵに対応する共有変数の全てが前記更新前の世代と一致しないかどうかを、複数の共有変数の値の列と所定の同期成立確認パターンとを比較することにより検出し、他ＣＰＵに対応する共有変数の全てが前記更新前の世代と一致しない状態となったときにバリア同期成立と判定する。

【００１０】より具体的には、並列計算機を構成する複数のＣＰＵ間でバリア同期をとる方法において、前記ＣＰＵに1対1に対応する共有変数の全てを、同じ同期世代を示す値に初期化する第1のステップと、バリア部分に実行が達した前記ＣＰＵのそれぞれにおいてバリア同期処理を実行する第2のステップとを含み、前記バリア同期処理は、（ａ）第1世代の次の世代を第2世代、第2世代の次の世代を第3世代、第3世代の次の世代を第1世代とする3世代のリング状遷移規則に従って、自ＣＰＵに対応する前記共有変数の値を、その値が示す世代の次の世代を示す値に更新するステップと、（ｂ）他ＣＰＵに対応する共有変数の全てが自ＣＰＵに対応する前記更新前の共有変数の値と一致しないという条件が成立するまで待ち合わせ、前記条件が成立したときバリア同期の成立と判定するステップとを含むことを特徴とする。

【００１１】また本発明は、前記ステップbが、前記更新前の共有変数の値を複数個並べた同期成立確認パターンと、他ＣＰＵに対応する共有変数の値を同じ個数並べたパターンとを比較する処理を含むことを特徴とする。ここで、同期成立確認パターンは合計3種類あり、各バリア同期処理中に適切なものを毎回作成して使用するようにしても良く、また予め作成され保存された3種類の前記同期成立確認パターンの中から適切なものを選択して使用するようにしても良い。

【００１２】また本発明は、前記共有変数のビット数を3ビットとし、前記ＣＰＵから1度でリード、ライトできる8×m（mは1以上の整数）ビットの記憶領域に8×m÷3の商に等しい個数の共有変数を格納し、前記同期成立確認パターンとして8×mビットのパターンを使用することにより、１度の比較で、8×m÷3の商に等しい数の共有変数の値が、所定の値でないかどうかを調べるようにしている。ここで、前記全ての共有変数は全ＣＰＵで共有される共有メモリ上に格納するようにしても良く、また、前記全ての共有変数を、他ＣＰＵから遠隔メモリアクセス可能な前記それぞれのＣＰＵのローカルメモリ上に格納するようにしても良い。この場合、一般の計算機では独立してアクセス可能な最小サイズは8ビット（１バイト）なので、共有変数を3ビットにすると、同じ1バイト中に複数のＣＰＵに対応する共有変数が含まれることになり、共有変数更新時のアクセス競合が問題となる。その解決方法として、1バイト単位で排他制御することも考えられるが、そうするとロック制御による処理速度の低下が起きるので、ロック処理を不要にするためには自ＣＰＵに対応する前記共有変数の値の更新をアトミックな論理演算を用いて行うことが望ましい。

【００１３】また本発明は、前記共有変数のビット数を8ビットとし、前記ＣＰＵから1度でリード、ライトできる8×r（rは2以上の整数）ビットの記憶領域にr個の共有変数を格納し、前記同期成立確認パターンとして8×rビットのパターンを使用することにより、１度

5

の比較で、r 個の共有変数の値が所定の値でないかどうかを調べるようにしている。ここで、前記全ての共有変数は全CPUで共有される共有メモリ上に格納するようにしても良く、また、前記全ての共有変数を、他CPUから遠隔メモリアクセス可能な前記それぞれのCPUのローカルメモリ上に格納するようにしても良い。この場合、共有変数を8ビット（1バイト）としているため、自CPUに対応する前記共有変数の値の更新はアトミックな論理演算を用いる必要はなく、またロック処理も不要である。

【0014】

【作用】次に、本発明の作用を説明する。なお、説明を簡単にするために、バリア同期をとるCPUは、CPUxとCPUyの2台とし、第1世代をA、第2世代をB、第3世代をCで示す。

【0015】まず、図1に示されるように、CPUx、CPUyに対応する共有変数を同じ同期世代、例えば第1世代Aを示す値に初期化する。その後、最初にCPUxの実行がバリア部分B1に達したとすると、バリア同期処理が実行される。まず、CPUxは、自CPUの共有変数の値を第1世代Aの次の世代である第2世代Bを示す値に更新する。次いで、CPUxは、他CPUyに対応する共有変数が前記更新前の共有変数の値が示す世代、つまりAでないという条件が成立するまで待ち合わせる。またCPUxに引き続きCPUyの実行がバリア部分B1に達すると、同じくバリア同期処理が実行され、まずCPUyは自CPUの共有変数の値を第1世代Aの次の世代である第2世代Bを示す値に更新する。この更新により、待ち合わせていたCPUxにおいて前記条件が成立し、CPUxはバリア同期が成立したと判定し、引き続く処理を実行する。また、CPUyが、他CPUxに対応する共有変数が前記更新前の共有変数の値が示す世代、つまりAでないという条件が成立したか否かを判定すると、既に成立しているので、待ち合わせることなく、バリア同期が成立したと判定し、引き続く処理を実行する。

【0016】CPUx、CPUyの実行が進み、次のバリア部分B2に達すると、バリア部分B1に達した場合と同様の処理が実行される。但し、前回のバリア部分B1の同期処理で世代がBに更新されているので、今回は更に次の世代である第3世代Cに変更される。同様に、更に次のバリア部分B3における同期処理では、共有変数は第3世代Cの次の世代である第1世代Aに変更される。

【0017】他方、図2はプロセスのディスパッチング等によりCPUyのバリア同期処理が共有変数を第2世代Bに更新した直後に中断した状態を示す。この場合、CPUyに対応する共有変数は第2世代Bに更新されているため、CPUxのバリア部分B1にかかるバリア同期処理においてバリア同期が成立したと判定され、引き

6

続く処理が実行される。そして、CPUxの実行が次のバリア部分B2に達すると、CPUxにおいて、自CPUに対応する共有変数が第3世代Cを示す値に更新され、他CPUyの共有変数の値が前記更新前の世代、つまり第2世代Bでなくまるまで待ち合わせる。

【0018】図3は、中断したCPUyのバリア同期処理が再開され、バリア部分B1にかかるバリア同期処理における同期成立の判定が行われている様子を示す。この同期判定では、CPUyは、他CPUxの共有変数が、自CPUの更新前の共有変数が示す世代、つまり第1世代Aでないか否かを調べる。CPUyの共有変数はバリア部分B2のバリア同期処理において第3世代Cに更新されているため、同期成立と判定され、CPUyは引き続く処理を実行する。そして、次のバリア部分B2に達して、CPUyが自CPUに対応する共有変数を第3世代Cに更新すると、待ち合わせ状態にあったCPUxにおいてバリア同期が成立したと判定される。

【0019】図2および図3を参照して説明したように、同期の検出が遅れたCPUyが存在しており、このCPUyがバリア部分B1の同期を検出しないうちに他のCPUxが次のバリア部分B2に到達してバリア同期処理を開始したとしても、CPUxで同期が成立するためには、同期の検出が遅れているCPUyが前回のバリア部分B1の同期を検出し、更に次のバリア部分B2に到達しなければ、先行しているCPUyのバリア同期は成立しない。このようなバリア同期は面バリア方式と呼ばれる。面バリア方式では、同期が二世代以上進んでしまうCPUが存在し得ないため、バリア同期の世代は3世代で必要充分である。これにより、各CPUに対応する共有変数のサイズを小さくできる。

【0020】また、同期が二世代以上進んでしまうCPUが存在し得ないため、同期の検出は、各CPUにおいて、全ての他CPUの共有変数が示す世代が、自CPUの更新後と同じ世代か或いは次の世代に一致しているか否かで行うことができる。そして、世代数は3世代なので、同一世代あるいは次の世代であることを検出することは前の世代ではないことを検出することと等価である。そこで、本発明では、他CPUに対応する共有変数の全てが自CPUに対応する更新前の共有変数の値でないという条件が成立しているか否かを調べるようにしており、更にその際、更新前の共有変数の値を複数個並べた同期成立確認パターンと、他CPUに対応する共有変数の値を同じ個数並べたパターンとを比較することにより、複数の共有変数のチェックを一括して行うようにしている。この結果、バリア同期が成立したか否かを少ない演算量で確認でき、バリア同期処理の高速化が図れる。

【0021】

【発明の実施の形態】次に本発明の実施の形態の例について図面を参照して詳細に説明する。

【００２２】図4を参照すると、本発明の一実施の形態は、並列計算機を構成するｎ＋1（ｎは1以上の整数）台のＣＰＵ1 −0 〜1 −nと、各ＣＰＵ1 −0 〜1 −nに1対1に対応する共有変数2 −0 〜2 −nを記憶する記憶部9とを備え、各ＣＰＵ1 −0 〜1 −nのそれぞれは、ＣＰＵ1 −0内にのみ図示するように、バリア同期処理に関連する手段として、共有変数初期化手段11、共有変数更新手段12、同期成立判定手段13および同期成立確認パターン生成手段14を有している。

【００２３】各共有変数2 −0 〜2 −nは、各ＣＰＵ1 −0 〜1 −nからアクセス可能な記憶部9上に設けられており、三世代の同期世代を表現するのに必要充分なサイズを有している。三世代の同期世代は、第1世代、第2世代、第3世代の3つであり、第1世代→第2世代→第3世代→第1世代というようなリング状の遷移規則に従って遷移せしめられる。

【００２４】各ＣＰＵ1 −0 〜1 −n中の共有変数初期化手段11は、例えば自ＣＰＵの起動時点に、自ＣＰＵに対応する共有変数を予めシステムで定められた初期の同期世代を示す値に初期化する手段である。

【００２５】共有変数更新手段12は、自ＣＰＵの実行がバリア部分に到達した際に、自ＣＰＵに対応する共有変数を更新する手段である。この更新は、前記リング状遷移規則に従って実行される。即ち、自ＣＰＵに対応する共有変数の値を、その値が示す世代の次の世代を示す値に更新する。

【００２６】同期成立判定手段13は、バリア同期が成立したか否かを判定する手段であり、他ＣＰＵに対応する共有変数の全てが自ＣＰＵに対応する前記更新前の共有変数の値と一致しないという条件が成立するまで待ち合わせ、前記条件が成立したときバリア同期の成立と判定する。

【００２７】同期成立確認パターン生成手段14は、同期成立判定手段13において前記条件が成立しているか否かを調べるために使用する同期成立確認パターンを生成する手段である。同期成立確認パターンは合計3種類ある。その1つは、第1世代を示す値を並べたパターン、他の1つは、第2世代を示す値を並べたパターン、最後の1つは第3世代を示す値を並べたパターンである。同期成立確認パターン生成手段14は、各バリア同期処理中に適切なものを毎回作成して同期成立判定手段13に提供するか、または、予め3種類の同期成立確認パターンを作成して保存しておき、各バリア同期処理中に適切なものを選択して同期成立判定手段13に提供する。

【００２８】図5は各ＣＰＵ1 −0 〜1 −nにおいて実行されるバリア同期に関連する処理の流れを示すフローチャートである。先ず、各ＣＰＵ1 −0 〜1 −nの共有変数初期化手段11は、自ＣＰＵに対応する共有変数2 −0 〜2 −nを予め定められた世代に初期化する（ステップＳ1）。どの世代に初期化するかは、全ＣＰＵ1 −0 〜1 −nで同じであれば良い。例えば全ＣＰＵ1 −0 〜1 −nの共有変数初期化手段11は、自ＣＰＵに対応する共有変数2 −0 〜2 −nを第1世代に初期化する。

【００２９】各ＣＰＵ1 −0 〜1 −n上のプログラムの実行が進行し、プログラム中に設定されたバリア同期点に到達すると、各ＣＰＵ1 −0 〜1 −nはバリア同期処理（ステップＳ2、Ｓ3）を実行する。まず、バリア同期点に到達したＣＰＵ1 −i（ｉ＝0 〜ｎ）の共有変数更新手段12は、自ＣＰＵに対応する共有変数2 −iを、その値が示す世代の次の世代を示す値に更新する（ステップＳ2）。即ち、共有変数2 −iが第1世代であれば第2世代に、第2世代であれば第3世代に、第3世代であれば第1世代に、それぞれ更新する。

【００３０】次にＣＰＵ1 −iの同期成立判定手段13は、他ＣＰＵに対応する共有変数の全てが自ＣＰＵに対応する前記更新前の共有変数の値と一致しないという条件が成立するか否かを調べ、成立していなければ当該条件が成立するまで待ち合わせ、前記条件が成立したときはバリア同期の成立と判定する（ステップＳ3）。前記条件が成立するか否かの判定は、より具体的には、同期成立確認パターン生成手段14で生成された同期成立確認パターンと複数の共有変数2 −0 〜2 −nの値の列とを比較することで実施する。

【００３１】例えば、ＣＰＵ1 −iの共有変数更新手段12が共有変数2 −iを第2世代に更新した場合、その1つ前の世代は第1世代なので、同期成立確認パターン生成手段14によって生成される3種類のパターンのうち第1世代を示す値を複数並べた同期成立確認パターンと、共有変数2 −0 〜2 −nのうちの同数の共有変数の値を並べたパターンとを比較する。また、共有変数2 −iを第3世代に更新した場合は、同期成立確認パターン生成手段14によって生成された第2世代を示す値を複数並べた同期成立確認パターンを使い、更に、共有変数2 −iを第1世代に更新した場合には第3世代を示す値を複数並べた同期成立確認パターンを使う。

【００３２】次に本発明の実施形態の実施例について図面を参照して詳細に説明する。

【００３３】（１）第1実施例

図6を参照すると、本実施例の並列計算機は、合計16台のＣＰＵ1 −0 〜1 −15がメモリコントローラ3経由で直接アクセス可能な共有メモリ4を備えており、この共有メモリ4上に各ＣＰＵ1 −0 〜1 −15に1対1に対応する共有変数2 −0 〜2 −15を格納する共有変数領域5を設けてある。また、メモリコントローラ3は、共有メモリ4に対するアトミックな論理演算を提供している。ここで、アトミックな論理演算とは、或るバイトのメモリデータをリードして、モディファイ（他データとのANDやOR）し、再度ライトするまでの処理が一体不可分な演算を意味する。

9

【００３４】共有変数2 -0 ～2 -15 のそれぞれは3 ビットであり、第1 世代は「００１」、第2 世代は「０１０」、第3 世代は「１００」で表現される。即ち、世代数に等しいビット数の共有変数を使用し、各ビットを各世代に1 対1 に対応付け、３ビットのうち、表現しようとする世代に対応するビットのみ“１”にする。

【００３５】各CPU1 -0 ～1 -15 の有するデータバス幅を３２ビットとし、各CPU1 -0 ～1 -15 からメモリコントローラ３を介して共有メモリ４に対し1 度でリード、ライトできるビット数（ワード）を３２ビットとすると、共有変数2 -0 ～2 -15 は、２個の３２ビットの整数型変数6 -0 、6 -1 に全て格納できる。つまり、整数型変数6 -0 の先頭より１０個の共有変数2 -0 ～2 -9 を格納し、次の整数型変数6 -1 の先頭より６個の共有変数2 -10 ～2 -15 を格納する。このとき、整数型変数6 -0 の未使用の2 ビット、整数型変数6 -1 の未使用の１４ビットには、それぞれ“０”を設定しておく。

【００３６】また、３種類の同期成立確認パターンは、本実施例では事前に生成しておいて共有メモリ４上の同期成立確認パターン領域8 に7 -0 ～7 -2 として格納しておく。同期成立確認パターン7 -0 は、３２ビットの整数型変数の先頭より「００１」の値を１０個並べ、最後の2 ビットを“０”としたパターン、同期成立確認パターン7 -1 は、３２ビットの整数型変数の先頭より「０１０」の値を１０個並べ、最後の2 ビットを“０”としたパターン、同期成立確認パターン7 -2 は、３２ビットの整数型変数の先頭より「１００」の値を１０個並べ、最後の2 ビットを“０”としたパターンである。

【００３７】このように、共有変数のビット数が3 ビットの場合、各CPUから1 度でリード、ライトできる共有メモリ４上の記憶領域を8 ×m（m は1 以上の整数）ビットとすると、１つの記憶領域に8 ×m÷3 の商に等しい個数の共有変数を格納し、同期成立確認パターンとして8 ×m ビットのパターンを使用する。

【００３８】図7 は本実施例の各CPU1 -0 ～1 -15 において実行されるバリア同期に関連する処理の流れを示すフローチャートである。なお、このようなバリア同期に関連する処理を司るバリア同期用プログラムは、CD-ROM、磁気ディスク、半導体メモリ等の機械読み取り可能な記録媒体K1 に記録されており、並列計算機の立ち上げ時等にメモリコントローラ３経由で各CPU1 -0 ～1 -15 に読み込まれ、各CPU1 -0 ～1 -15 の動作を制御する。

【００３９】先ず、各CPU1 -0 ～1 -15 の共有変数初期化手段１１は、自CPUに対応する共有変数2 -0 ～2 -15 を予め定められた世代、例えば第1 世代を示す値「００１」に初期化する（ステップS１１）。次に各CPU1 -0 ～1 -15 の同期成立確認パターン生成手段１４は、メモリコントローラ３を介して共有メモリ４の同期成立確認パターン領域8 をアクセスし、既に他のCPUが3 種類の同期成立確認パターンを生成して格納してあるか否かを調べる（ステップS１２）。未だ格納されていない場合に限り、同期成立確認パターン生成手段１４は、３種類の同期成立確認パターン7 -0 ～7 -2 を生成してメモリコントローラ３経由で同期成立確認パターン領域8 に格納する（ステップS１３）。

10

【００４０】各CPU1 -0 ～1 -15 上のプログラムの実行が進行し、プログラム中に設定されたバリア同期点に到達すると、各CPU1 -0 ～1 -15 はバリア同期処理（ステップS１４、S１５）を実行する。即ち、バリア同期点に到達したCPU1 -i （i =0 ～15）の共有変数更新手段１２は、先ず、自CPUに対応する共有変数2 -i を、その値が示す世代の次の世代を示す値に更新する（ステップS１４）。具体的には、メモリコントローラ３経由で共有メモリ４上の自CPUに対応する共有変数2 -i を含む３２ビットの整数型変数をリードして、共有変数2 -i の値を内部レジスタに保存した後、先ず、アトミックな論理和（OR）演算によって、現在の共有変数2 -i の値と数値「１１１」との論理和で共有メモリ４上の元の共有変数2 -i を更新し、次いで、アトミックな論理積（AND）演算によって、現在の共有変数2 -i の値（つまり「１１１」）と次同期世代を示す値との論理積で共有メモリ４上の元の共有変数2 -i を更新する。

【００４１】例えば、共有変数2 -i が第1 世代を示す「００１」になっている状態でバリア同期処理を開始したとき、アトミックな論理和によって共有変数2 -i の値が「１１１」に更新され、次にアトミックな論理積によって「１１１」と次同期世代を示す値「０１０」との論理積である「０１０」が元の共有変数2 -i に設定される。ここで、或る1 バイトデータA中の共有変数2 -i に対応する3 ビットだけを「１１１」に更新する論理和演算は、共有変数2 -i に対応する３ビットを「１１１」とし他は“０”とした1 バイトデータBと前記1 バイトデータAとの論理和をとることで可能である。また、この論理和演算で更新された1 バイトデータC中の共有変数2 -i に対応する3 ビットの値「１１１」だけを所定の値「０１０」に更新する論理積演算は、共有変数2 -i に対応する３ビットを「０１０」とし他のビットは元の1 バイトデータCの値とした1 バイトデータDと前記1 バイトデータCとの論理積をとることで可能である。なお、論理和演算と論理積演算との2 種類の演算を使用するのは、一般の計算機にはビットオペレーション演算機能が備わっていないからである。また、最初に「１１１」に変更するのは、アトミックな論理和とアトミックな論理積との間に、他CPUから当該共有変数2 -i の参照があっても同期成立と判定されないようにするためである。

【００４２】共有変数2 -i の更新後、同期成立判定手

11

段13は、自CPUに対応する共有変数2 -i も含め共有変数領域5 上の全ての共有変数の値が自CPUに対応する前記更新後の共有変数の値より1つ前の世代を示す値でないという条件が成立するか否かを調べ、成立していなければ当該条件が成立するまで待ち合わせ、前記条件が成立したときはバリア同期の成立と判定する(ステップS15)。具体的には、まず同期成立確認パターン生成手段14は、同期成立確認パターン領域8 に格納された3 種類の同期成立確認パターンの内、前記内部レジスタに保存された更新前の同期世代の値を格納した同期成立確認パターンをメモリコントローラ3 経由で共有メモリ4から取得する。次いで、同期成立判定手段13は、メモリコントローラ3 経由で共有変数領域5 から32ビットの整数型変数6 -0、6 -1を1つずつ読み込み、前記取得された同期成立確認パターンとの論理積(AND)を求める。全ての整数型変数6 -0、6 -1と同期成立確認パターンとの論理積の値が0 であれば、バリア同期成立と判断し、そうでなければ、全ての整数型変数6 -0、6 -1と同期成立確認パターンとの論理積の値が0 になるまで待ち合わせを行う。

【0043】例えばバリア同期処理を行っているCPU1 -i をCPU1 -0とし、自CPUに対応する共有変数2 -0を第2 世代を示す値「010」に更新した後、1つ前の同期世代を示す値「001」を格納した同期成立確認パターン7 -0と共有変数領域5 中の32ビットの整数型変数6 -0との論理積をとっている様子を図8に示す。この状態では、CPU1 -1の共有変数2 -1の値が「010」になっているため、論理積の結果は0にならない。このため、CPU1 -0は論理積の結果が0になるまで待ち合わせを行うことになる。

【0044】以上のように本実施例によれば、同期の検出時に1 回の比較処理(AND演算)で10個分のCPUの共有変数のチェックが可能となり、その分、バリア同期を高速に実行することが可能となる。勿論、CPUが64ビットのデータバスを備え、64ビット長の整数型変数を利用できるなら、より効率良く同期の検出を行うことが可能である。

【0045】(2)第2 実施例

図9を参照すると、本実施例の並列計算機は、合計16台のCPU1 -0～1 -15が共有メモリを持たず、その代わりに独立したローカルメモリ11 -0～11 -15と、他CPUに備わるローカルメモリの値をクロスバスイッチ等のノード間ネットワーク13 経由で参照および定義する手段を有する遠隔ノードメモリアクセス装置12 -0～12 -15とを備えており、各ローカルメモリ11 -0～11 -15上に各CPU1 -0～1 -15に1対1に対応する共有変数2 -0～2 -15を格納する共有変数領域5、および3種類の同期成立確認パターン7 -0～7 -2を格納する同期成立確認パターン領域8を設けてある(図ではローカルメモリ11 -0内にのみ図示してある)。また、各遠隔ノードメモリアクセス装置12 -0～12 -15は、更に、指定されたCPUの指定されたローカルメモリに対してアトミックな論理演算を行う機能と、指定されたCPUの組に対して上記アトミック演算やデータの定義を同報(マルチキャスト)する機能とを備えている。

12

【0046】本実施例では、共有メモリが存在しないため、全CPU1 -0～1 -15の共有変数2 -0～2 -15が、全CPUのローカルメモリ11 -0～11 -15にそれぞれ配置され、各CPUはバリア同期のたびに、全CPUのローカルメモリ1 -0～11 -15上に存在する自CPU対応の共有変数を遠隔ノードメモリアクセス装置12 -0～12 -15を介して更新し、バリア同期点に達していることを他のCPUに通知する。

【0047】また本実施例においても、前記第1 の実施例と同様に、共有変数2 -0～2 -15のそれぞれは3ビットであり、第1 世代は「001」、第2 世代は「010」、第3 世代は「100」で表現される。また、各CPU1 -0～1 -15の有するデータバス幅を32ビットとし、各CPU1 -0～1 -15からローカルメモリに対し1 度でリード、ライトできるビット数(ワード)を32ビットとしている。このため、共有変数2 -0～2 -15は、第1 の実施例と同様にして、2個の32ビットの整数型変数6 -0、6 -1に全て格納される。更に、3種類の同期成立確認パターンは、本実施例でも事前に生成しておいてローカルメモリ11 -0～11 -15上の同期成立確認パターン領域8 に7 -0～7 -2として格納しておく。同期成立確認パターン7 -0～7 -2は第1 の実施例と同様のものである。

【0048】図10は本実施例の各CPU1 -0～1 -15において実行されるバリア同期に関連する処理の流れを示すフローチャートである。なお、このようなバリア同期に関連する処理を司るバリア同期用プログラムは、CD-ROM、磁気ディスク、半導体メモリ等の機械読み取り可能な記録媒体K2 に記録されており、並列計算機の立ち上げ時等に各CPU1 -0～1 -15に読み込まれ、各CPU1 -0～1 -15の動作を制御する。

【0049】先ず、各CPU1 -0～1 -15の共有変数初期化手段11は、自CPUのローカルメモリ11 -i の共有変数領域5 上の全ての共有変数2 -0～2 -15を予め定められた世代、例えば第1 世代を示す値「001」に初期化する(ステップS21)。

【0050】次に各CPU1 -0～1 -15の同期成立確認パターン生成手段14は、3種類の同期成立確認パターン7 -0～7 -2を生成し、自ローカルメモリ11 -0～11 -15上の同期成立確認パターン領域8 に格納する(ステップS22)。

【0051】各CPU1 -0～1 -15上のプログラムの実行が進行し、プログラム中に設定されたバリア同期

点に到達すると、各ＣＰＵ1 −0 ～1 −１５はバリア同期処理（ステップＳ２３、Ｓ２４）を実行する。まず、バリア同期点に到達したＣＰＵ1 −i の共有変数更新手段１２は、自ＣＰＵに対応する共有変数2 −i を、その値が示す世代の次の世代を示す値に更新する（ステップＳ２３）。具体的には、自ローカルメモリ１１−i 上の自ＣＰＵに対応する共有変数2 −i を含む３２ビットの整数型変数をリードして、共有変数2 −i の値を内部レジスタに保存した後、先ず、アトミックな論理和（ＯＲ）演算によって、自ローカルメモリ１１−i 上の現在の共有変数2 −i の値と数値「１１１」との論理和で自ローカルメモリ１１−i 上の元の共有変数2 −i を更新すると共に、遠隔ノードメモリアクセス装置１２−i の有するアトミックな論理和（ＯＲ）演算機能を使って、他ＣＰＵのローカルメモリ上の共有変数2 −i の値と数値「１１１」との論理和でそのローカルメモリ上の元の共有変数2 −i を更新する。次いで、アトミックな論理積（ＡＮＤ）演算によって、自ＣＰＵ1 −i のローカルメモリ１１−i 上の現在の共有変数2 −i の値（つまり「１１１」）と次同期世代を示す値との論理積で自ローカルメモリ１１−i 上の元の共有変数2 −i を更新すると共に、遠隔ノードメモリアクセス装置１２−i の有するアトミックな論理積（ＡＮＤ）演算機能を使って、全ての他ＣＰＵのローカルメモリ上の共有変数2 −i の値（つまり「１１１」）と次同期世代を示す値との論理積でそのローカルメモリ上の元の共有変数2 −i を更新する。

【００５２】共有変数2 −i の更新後、同期成立判定手段１３は、自ＣＰＵに対応する共有変数2 −i も含め自ローカルメモリ１１−i の共有変数領域5 上の全ての共有変数の値が自ＣＰＵに対応する前記更新前の共有変数の値と一致しないという条件が成立するか否かを調べ、成立していなければ当該条件が成立するまで待ち合わせ、前記条件が成立したときはバリア同期の成立と判定する（ステップＳ２４）。具体的には、まず同期成立確認パターン生成手段１４は、同期成立確認パターン領域8 に格納された3 種類の同期成立確認パターンの内、前記内部レジスタに保存された更新前の同期世代の値を格納した同期成立確認パターンを取得する。次いで、同期成立判定手段１３は、共有変数領域5 から３２ビットの整数型変数6 −0 、6 −1 を1 つずつ読み込み、前記取得された同期成立確認パターンとの論理積（ＡＮＤ）を求める。全ての整数型変数6 −0 、6 −1 と同期成立確認パターンとの論理積の値が0 であれば、バリア同期成立と判断し、そうでなければ、全ての整数型変数6 −0 、6 −1 と同期成立確認パターンとの論理積の値が0 になるまで待ち合わせを行う。

【００５３】以上のように本実施例によれば、第1 の実施例と同様に、同期の検出時に1 回の比較処理（ＡＮＤ演算）で１０個分のＣＰＵの共有変数のチェックが可能となり、その分、バリア同期を高速に実行することが可能となる。

【００５４】（３）第3 実施例

図１１を参照すると、本実施例の並列計算機は、第2 の実施例と同様に合計１６台のＣＰＵ1 −0 ～1 −１５が共有メモリを持たず、その代わりに独立したローカルメモリ１１−0 ～１１−１５と、他ＣＰＵに備わるローカルメモリの値をノード間ネットワーク１３経由で参照および定義する手段を有する遠隔ノードメモリアクセス装置１２−0 ～１２−１５とを備えており、各ローカルメモリ１１−0 ～１１−１５上に各ＣＰＵ1 −0 ～1 −１５に1 対1 に対応する共有変数2 −0 ～2 −１５を格納する共有変数領域5 、および3 種類の同期成立確認パターン7 −0 ～7 −2 を格納する同期成立確認パターン領域8 を設けてある（図ではローカルメモリ１１−0 内にのみ図示してある）。また、各遠隔ノードメモリアクセス装置１２−0 ～１２−１５は、ローカルメモリに対してアトミックな論理演算を行う機能は有していないが、指定されたＣＰＵの組に対してデータの定義を同報（マルチキャスト）する機能は備えている。

【００５５】本実施例では、共有メモリが存在しないため、第2 の実施例と同様に、全ＣＰＵ1 −0 ～1 −１５の共有変数2 −0 ～2 −１５が、全ＣＰＵのローカルメモリ１１−0 ～１１−１５にそれぞれ配置され、各ＣＰＵはバリア同期のたびに、全ＣＰＵのローカルメモリ１−0 ～１１−１５上に存在する自ＣＰＵ対応の共有変数を遠隔ノードメモリアクセス装置１２−0 ～１２−１５を介して更新し、バリア同期点に達していることを他のＣＰＵに通知する。

【００５６】また本実施例においては、前記第1 および第2 の実施例と異なり、共有変数2 −0 ～2 −１５のそれぞれは8 ビット（１バイト）であり、第1 世代は「００００００１」（１６進数で0ｘ１）、第2 世代は「０００００１０」（１６進数で0ｘ２）、第3 世代は「０００００１００」（１６進数で0ｘ４）で表現される。また、各ＣＰＵ1 −0 ～1 −１５の有するデータバス幅を３２ビットとし、各ＣＰＵ1 −0 ～1 −１５からローカルメモリに対し1 度でリード、ライトできるビット数（ワード）を３２ビット（４バイト）としている。更に、各ＣＰＵ1 −0 ～1 −１５および各遠隔ノードメモリアクセス装置１２−0 ～１２−１５は、バイトアクセス機能を備えている。また、共有変数2 −0 ～2 −１５は、４個の３２ビットの整数型変数6 −0 ～6 −3 に全て格納される。なお、若し、最後の整数型変数に未使用ビットができた場合には、その部分は“０”に設定しておく。

【００５７】更に、３種類の同期成立確認パターンは、本実施例でも事前に生成しておいてローカルメモリ１１−0 ～１１−１５上の同期成立確認パターン領域8 に7 −0 ～7 −2 として格納しておく。同期成立確認パター

15

ン7 −0は、３２ビットの整数型変数の先頭より「０ｘ１」の値を4個並べたパターン、同期成立確認パターン7 −1は、３２ビットの整数型変数の先頭より「０ｘ２」の値を4個並べたパターン、同期成立確認パターン7 −2は、３２ビットの整数型変数の先頭より「０ｘ４」の値を4個並べたパターンである。

【００５８】図１２は本実施例の各ＣＰＵ1 −0 ～1 −１５において実行されるバリア同期に関連する処理の流れを示すフローチャートである。なお、このようなバリア同期に関連する処理を司るバリア同期用プログラムは、ＣＤ−ＲＯＭ、磁気ディスク、半導体メモリ等の機械読み取り可能な記録媒体Ｋ３に記録されており、並列計算機の立ち上げ時等に各ＣＰＵ1 −0 ～1 −１５に読み込まれ、各ＣＰＵ1 −0 ～1 −１５の動作を制御する。

【００５９】先ず、各ＣＰＵ1 −0 ～1 −１５の共有変数初期化手段１１は、自ＣＰＵのローカルメモリ上の共有変数領域5にある全ての共有変数2 −0 ～2 −１５を予め定められた世代、例えば第1世代を示す値「０ｘ１」に初期化する（ステップＳ３１）。

【００６０】次に各ＣＰＵ1 −0 ～1 −１５の同期成立確認パターン生成手段１４は、３種類の同期成立確認パターン7 −0 ～7 −2を生成し、自ローカルメモリ１１ −0 ～１１ −１５上の同期成立確認パターン領域8に格納する（ステップＳ３２）。

【００６１】各ＣＰＵ1 −0 ～1 −１５上のプログラムの実行が進行し、プログラム中に設定されたバリア同期点に到達すると、各ＣＰＵ1 −0 ～1 −１５はバリア同期処理（ステップＳ３３、Ｓ３４）を実行する。まず、バリア同期点に到達したＣＰＵ1 −i の共有変数更新手段１２は、自ＣＰＵに対応する共有変数2 −i を、その値が示す世代の次の世代を示す値に更新する（ステップＳ３３）。具体的には、自ローカルメモリ１１ −i 上の自ＣＰＵに対応する共有変数2 −i を含む３２ビットの整数型変数をリードして、共有変数2 −i の値を内部レジスタに保存した後、自ローカルメモリ１１ −i 上の現在の共有変数2 −i の値をバイトアクセス機能により次の世代を示す値に更新すると共に、遠隔ノードメモリアクセス装置１２ −i の有するバイトアクセス機能を使って、他ＣＰＵのローカルメモリ上の共有変数2 −i の値を次の世代を示す値に更新する。

【００６２】全ローカルメモリ１１ −0 ～１１ −１５上の共有変数2 −i の更新後、同期成立判定手段１３は、自ＣＰＵに対応する共有変数2 −i も含め自ローカルメモリ１１ −i の共有変数領域5上の全ての共有変数の値が自ＣＰＵに対応する前記更新前の共有変数の値と一致しないという条件が成立するか否かを調べ、成立していなければ当該条件が成立するまで待ち合わせ、前記条件が成立したときはバリア同期の成立と判定する（ステップＳ３４）。具体的には、まず同期成立確認パターン生

16

成手段１４は、同期成立確認パターン領域8に格納された3種類の同期成立確認パターン7 −0 ～7 −2の内、前記内部レジスタに保存された更新前の同期世代の値を格納した同期成立確認パターンを取得する。次いで、同期成立判定手段１３は、自ローカルメモリ１１ −i の共有変数領域5から３２ビットの整数型変数6 −0、6 −1を1つずつ読み込み、前記取得された同期成立確認パターンとの論理積（ＡＮＤ）を求める。全ての整数型変数6 −0、6 −1と同期成立確認パターンとの論理積の値が0であれば、バリア同期成立と判断し、そうでなければ、全ての整数型変数6 −0、6 −1と同期成立確認パターンとの論理積の値が0になるまで待ち合わせを行う。

【００６３】以上のように本実施例によれば、同期の検出時に1回の比較処理（ＡＮＤ演算）で4個分のＣＰＵの共有変数のチェックが可能となり、その分、バリア同期を高速に実行することが可能となる。

【００６４】以上本発明の実施の形態および実施例について説明したが、本発明は以上の実施の形態および実施例に限定されず、例えば以下に述べるような各種の付加変更が可能である。

【００６５】並列計算機を構成する全ＣＰＵ間でバリア同期をとる場合について説明したが、並列計算機を構成する任意の複数のＣＰＵ間でのバリア同期に対して本発明は適用可能である。つまり、バリア同期に参加するＣＰＵが並列計算機を構成する全ＣＰＵの一部である場合、バリア同期に参加する全ＣＰＵ間で本発明によるバリア同期方法を実施すれば良い。

【００６６】複数のＣＰＵで共有される共有メモリを有する並列計算機が、アトミックな論理演算機能を有していない場合には、第3の実施例と同様に共有変数のサイズを1バイトとし、バイトアクセス機能によって共有変数を定義、参照する構成とすることができる。

【００６７】

【発明の効果】以上説明したように本発明によれば以下のような効果が得られる。

【００６８】各ＣＰＵに対応するバリア同期用の共有変数のサイズを小さくできる。その理由は、共有変数の値を3世代のリング状遷移規則に従って更新しており、第1世代、第2世代、第3世代の合計3世代の同期世代を表現できるだけのビット数で必要充分だからである。

【００６９】バリア同期が成立したか否かを少ない演算量で確認でき、バリア同期処理の高速化が可能である。その理由は、比較対象となる世代の値を複数個並べた同期成立確認パターンと、各ＣＰＵに対応する共有変数の値を同じ個数並べたパターンとを比較することにより、複数のＣＰＵの共有変数の値を一括してチェックしているからである。

【図面の簡単な説明】

【図1】本発明の作用の説明図である。

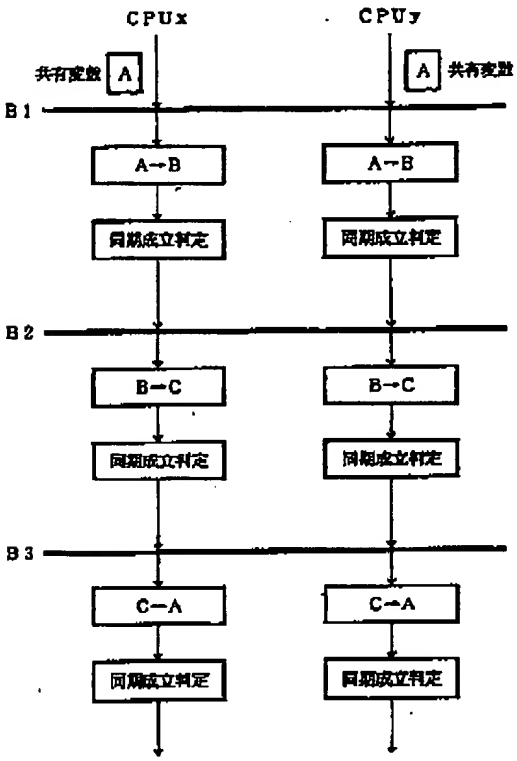
17

【図2】本発明の作用の説明図である。
【図3】本発明の作用の説明図である。
【図4】本発明の一実施の形態の構成例を示すブロック図である。
【図5】本発明の一実施の形態におけるバリア同期に関連する処理の流れを示すフローチャートである。
【図6】本発明の第1の実施例の構成例を示すブロック図である。
【図7】本発明の第1の実施例におけるバリア同期に関連する処理の流れを示すフローチャートである。
【図8】同期検出時の処理説明図である。
【図9】本発明の第2の実施例の構成例を示すブロック図である。
【図10】本発明の第2の実施例におけるバリア同期に関連する処理の流れを示すフローチャートである。
【図11】本発明の第3の実施例の構成例を示すブロック図である。

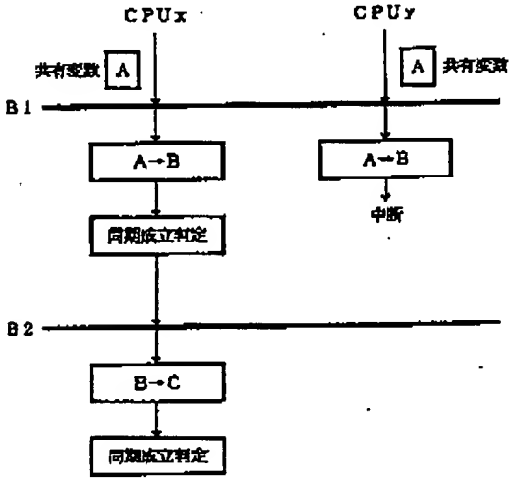
18

【図12】本発明の第3の実施例におけるバリア同期に関連する処理の流れを示すフローチャートである。
【符号の説明】
1-0～1-n …CPU
2-0～2-n …共有変数
3 …メモリコントローラ
4 …共有メモリ
5 …共有変数領域
6-0～6-3 …共有変数を格納する32ビットの整数型変数
7-0～7-2 …3種類の同期成立確認パターン
8 …同期成立確認パターン領域
9 …記憶部
11-0～11-15 …ローカルメモリ
12-0～12-15 …遠隔ノードメモリアクセス装置
13 …ノード間ネットワーク
K1～K3 …記録媒体

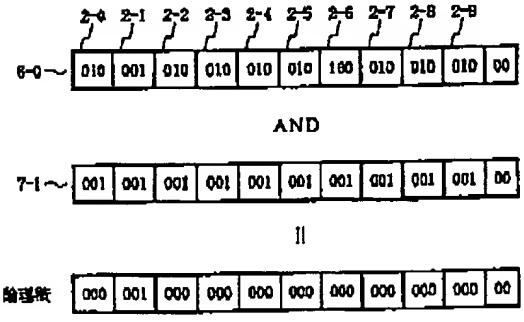
【図1】



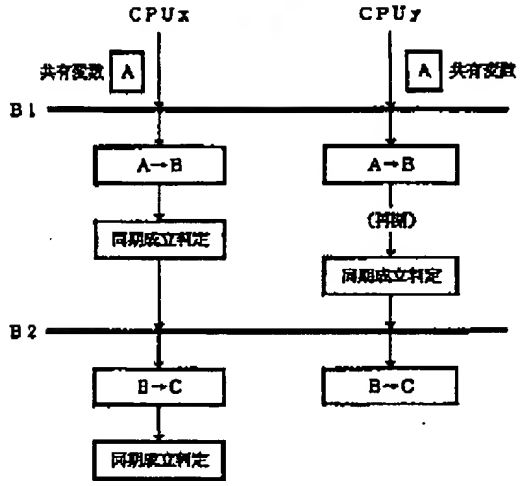
【図2】



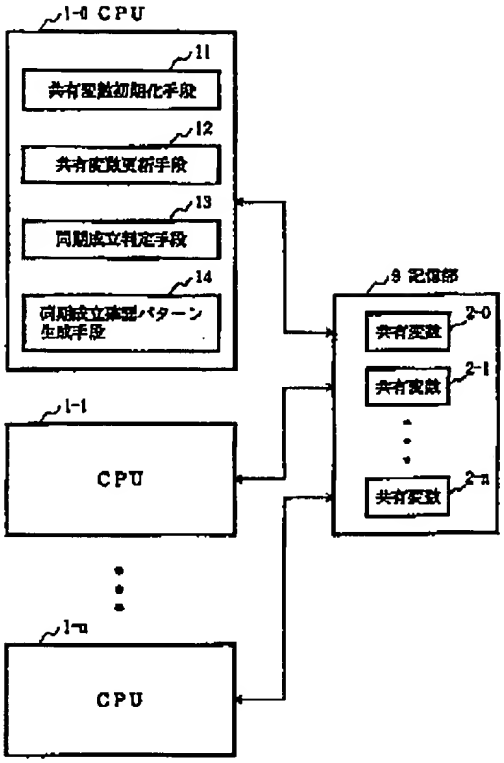
【図8】



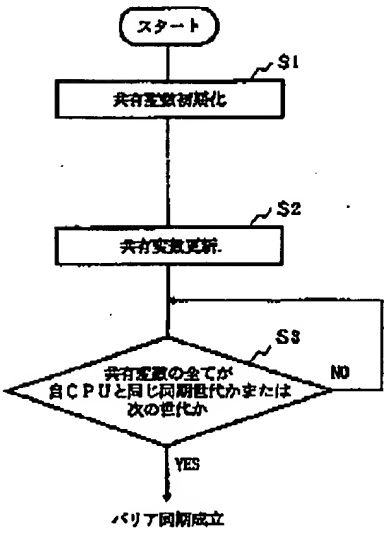
【図3】


CPUx
CPUy
共有変数
A
共有変数
B1
A→B
A→B
同期成立判定
(待機)
同期成立判定
B2
B→C
B→C
同期成立判定

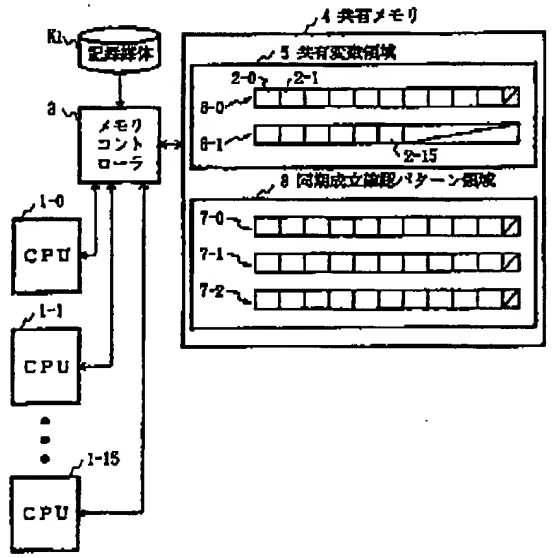
【図4】


1-0 CPU
11
共有変数初期化手段
12
共有変数更新手段
13
同期成立判定手段
14
同期成立論理パターン生成手段
9 記憶部
2-0
共有変数
2-1
共有変数
2-n
共有変数
1-1
CPU
1-n
CPU

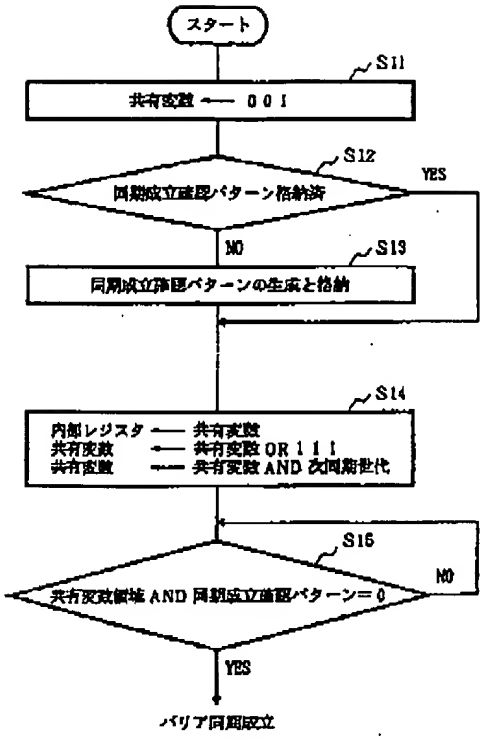
【図5】


スタート
S1
共有変数初期化
S2
共有変数更新
S3
共有変数の全てが自CPUと同じ同期世代かまたは次の世代か
NO
YES
バリア同期成立

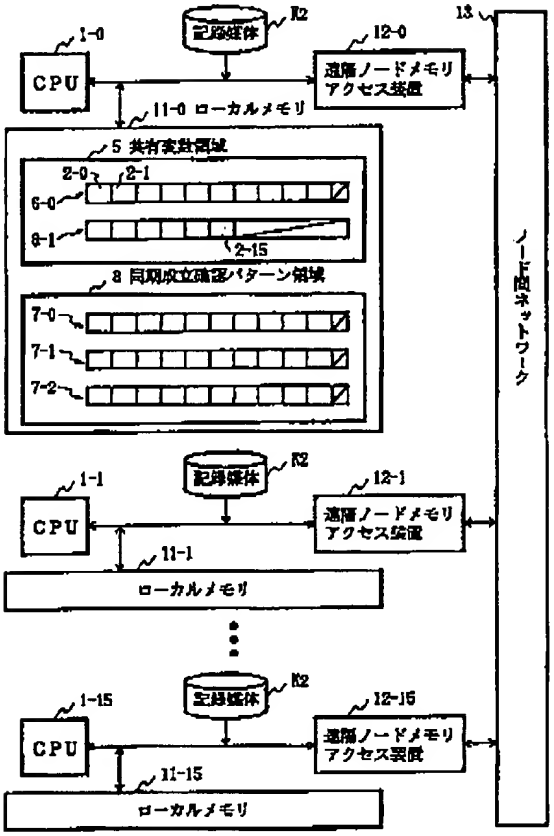
【図6】


4 共有メモリ
5 共有変数領域
2-0
2-1
8-0
8-1
2-15
8 同期成立論理パターン領域
7-0
7-1
7-2
K1
記憶媒体
3
メモリコントローラ
1-0
CPU
1-1
CPU
1-15
CPU

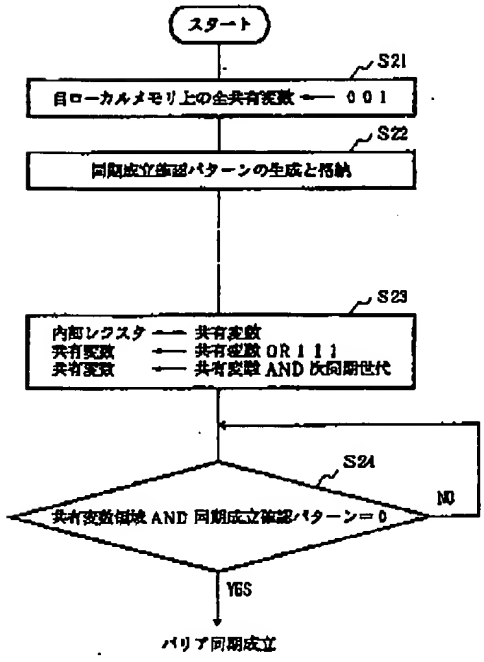
【図7】


スタート
S11
共有変数 ← 0 0 1
S12
同期成立確認パターン格納済
YES
NO
S13
同期成立確認パターンの生成と格納
S14
内部レジスタ ← 共有変数
共有変数 ← 共有変数 OR 1 1 1
共有変数 ← 共有変数 AND 次同期世代
S15
共有変数領域 AND 同期成立確認パターン=0
NO
YES
バリア同期成立

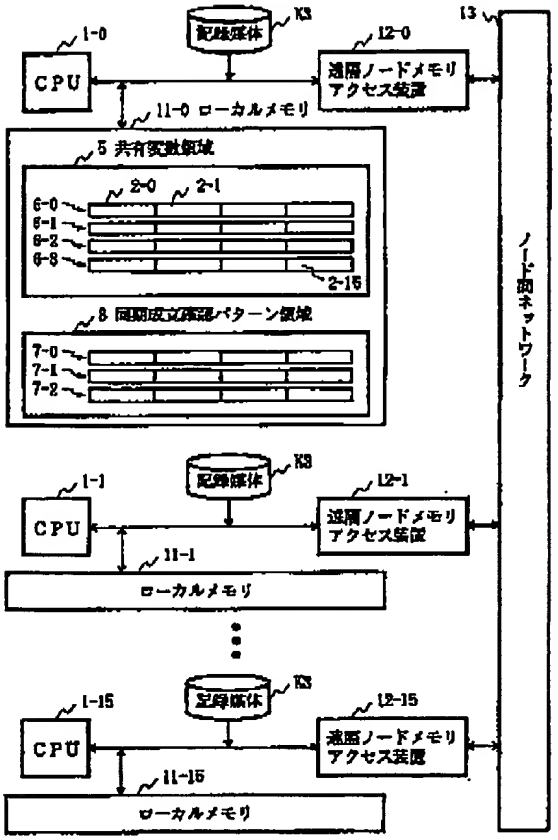
【図9】


1-0
CPU
M2
記憶媒体
12-0
通信ノードメモリアクセス装置
13
ノード間ネットワーク
11-0 ローカルメモリ
5 共有変数領域
2-0
2-1
6-0
6-1
2-15
8 同期成立確認パターン領域
7-0
7-1
7-2
1-1
CPU
M2
記憶媒体
12-1
通信ノードメモリアクセス装置
11-1
ローカルメモリ
1-15
CPU
M2
記憶媒体
12-15
通信ノードメモリアクセス装置
11-15
ローカルメモリ

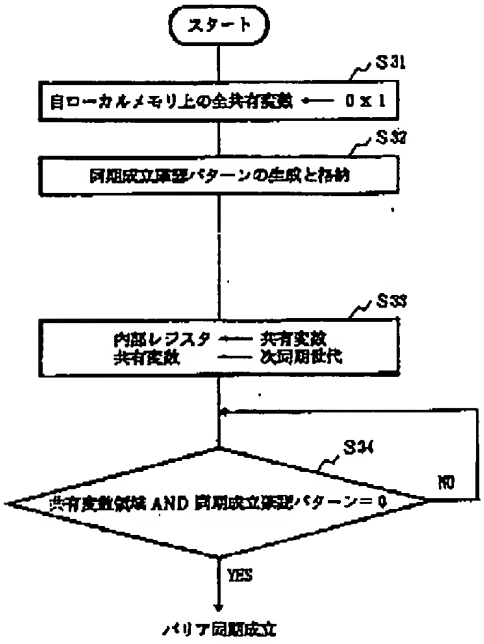
【図10】


スタート
S21
自ローカルメモリ上の全共有変数 ← 0 0 1
S22
同期成立確認パターンの生成と格納
S23
内部レジスタ ← 共有変数
共有変数 ← 共有変数 OR 1 1 1
共有変数 ← 共有変数 AND 次同期世代
S24
共有変数領域 AND 同期成立確認パターン=0
NO
YES
バリア同期成立

【図11】


CPU
1-0
記録媒体
K3
遠隔ノードメモリアクセス装置
12-0
13
ノード間ネットワーク
11-0 ローカルメモリ
5 共有変数領域
2-0
2-1
6-0
6-1
6-2
6-8
2-15
8 同期成立確認パターン領域
7-0
7-1
7-2
CPU
1-1
記録媒体
K3
遠隔ノードメモリアクセス装置
12-1
11-1
ローカルメモリ
CPU
1-15
記録媒体
K3
遠隔ノードメモリアクセス装置
12-15
11-15
ローカルメモリ

【図12】


スタート
S31
自ローカルメモリ上の全共有変数 ← 0 x 1
S32
同期成立確認パターンの生成と格納
S33
内部レジスタ ← 共有変数
共有変数 ← 次同期世代
S34
共有変数領域 AND 同期成立確認パターン= 0
NO
YES
バリア同期成立